**What Is Name Decoration?**

How does C++ keep track of which overloaded function is which? It assigns these functions a secret identity. When you use the editor of your C++ development tool to write and compile programs, your C++ compiler performs a bit of magic on your behalf—known as *name decoration* or *name mangling*—through which each function name is encrypted based on the formal parameter types specified in the function's prototype. Consider the following undecorated function prototype:

```
long MyFunctionFoo(int, float);
```

This format is fine for us humans; we know that the function accepts two arguments of type `int` and `float`, and returns a value of type `long`. For its own use, the compiler documents this interface by transforming the name into an internal representation of a more unsightly appearance, perhaps something like this:

```
?MyFunctionFoo@@YAXH@Z
```

The apparent gibberish decorating the original name (or mangling it, depending upon your attitude) encodes the number and types of parameters. A different function signature would have resulted in a different set of symbols being added, and different compilers would use different conventions for their efforts at decorating.

**Function Templates**

Contemporary C++ compilers implement one of the newer C++ additions, *function templates*. Function templates are a generic function description; that is, they define a function in terms of a generic type for which a specific type, such as `int` or `double`, can be substituted. By passing a type as a parameter to a template, you cause the compiler to generate a function for that particular type. Because templates let you program in terms of